Vol.2019-HPC-170 No.4
2019/7/24

InfiniBand verbsとDMA転送を用いたFPGAクラスター向けソフトウェア ブリッジド データ転送

宮島 敬明[1,a)]　平尾 智也[2]　宮元 直也[2]　孫 正道[2]　佐野 健太郎[1]

概要：
高性能計算の分野では Field Programmable Gate Array (FPGA) を搭載したヘテロジニアスシステムが注目を集めつつある。FPGA ボードを CPU に接続したアクセラレータとしてシステムを構築する場合、ネットワーク部を含めると複数の構成が考えられる。FPGA ボード上のメモリから別のホスト CPU 上のメモリへの実効データ転送帯域は、構成を決める上での重要な要素の一つである。最適な構成を決定するためには、ノードを跨いだデータ転送機能の実装と実効データ転送帯域の定量的な評価が必要である。本稿では 2 つの FPGA 搭載ノードを用いて、ノードを跨いだ FPGA へのデータ転送手法の提案と評価を行う。2 つのノードは 100Gbps InfiniBand HCA で接続されている。また、ノード内の FPGA とホスト CPU は PCIe Gen3 x8 で接続されている。提案するデータ転送機能は、我々がソフトウェアブリッジドデータ転送と呼ぶ、Direct Memory Access (DMA) 機能と InfiniBand verbs によるノード間データ転送をパイプライン的に組み合わせたものである。評価の結果、別ホストの CPU メモリから FPGA メモリでは 6379.1MB/s (81.0 %)、FPGA メモリから別ホストの CPU メモリでは 5696.1 MB/s (72.3 %)、ノードを往復するでは 4732.0 MB/s (60.1%) となった。これは経路全体のボトルネックである PCIe Gen3 x8 の理論ピーク性能に対し、81.0 %、72.3 %、60.1%である。

1. はじめに

近年、FPGA を搭載した大規模なヘテロジニアスシステムが登場してきている [4] [2] [7]。これらは、ネットワークと FPGA ボードに注目した場合、図 1 と 2 に示すような大まかに 2 つの構成に分類可能である。前者は、ホスト CPU と FPGA で別々に専用ネットワークを持つ。各ネットワークがトーラスの様な直結網であればネットワーク性能を最大限に利用できるため、これを性能指向構成と呼ぶ。この構成は、対象アプリケーションや通信パターンが事前に分かる場合に非常に有効である。後者は、ホスト CPU と FPGA が単一のネットワークを共有する。FPGA とホスト CPU がスイッチなどを介して自由かつ動的にデータを転送できる柔軟な構成のため、柔軟性指向構成と呼ぶ。FPGA にはデータ転送用のプロトコル変換や管理のために、CPU の様なコントローラが必要になる。この構成は、1 つのホスト CPU が複数の FPGA に直接アクセスできるため、複数の FPGA を利用・管理しやすい。運用上の利点として、1 つのネットワークが故障しても別のネットワークの障害とならないため、対障害性も前者の構成に比べて高い。また、アプリケーションによっては FPGA を全く

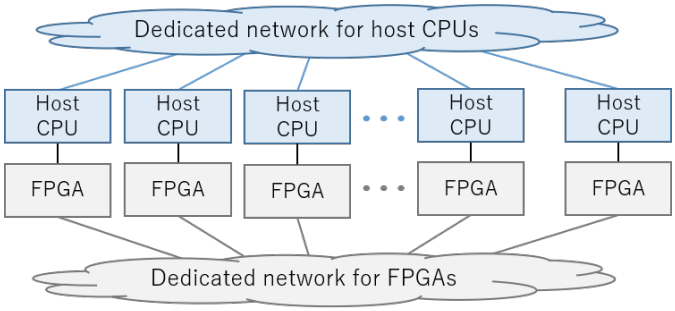

図 1: 性能指向型の構成：FPGA 部とホスト CPU 部がそれぞれ専用ネットワークを持つ

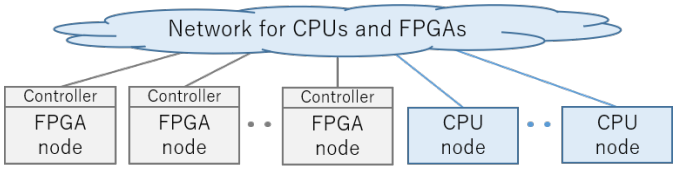

図 2: 柔軟性指向型の構成：FPGA 部とホスト CPU 部が共有のネットワークを持つ

[1] 理化学研究所 計算科学研究センタープロセッサ研究チーム
兵庫県神戸市中央区港島南町 6-3-5
[2] 株式会社フィックスターズ
東京都品川区大崎 1-11-1 ゲートシティ大崎ウエストタワー 18 階
[a)] takaaki.miyajima@riken.jp

利用しないものと複数利用するものがあり、後者の構成であれば資源を有効活用しやすい。しかし、FPGA へアクセスするには、スイッチとホスト CPU を介するため通信性能が問題となることが予想される。

また、将来の FPGA クラスタでは、FPGA 間のデータ通信やタスクマイグレーションが発生すると考えられる。この場合、コンフィグレーションデータや計算データなどの大容量のデータを別ノードの FPGA へ転送する必要がでてくるため、次の 2 種類のデータ転送機能が必須となる。

“A”:ノード内データ通信機能：FPGA メモリからホスト CPU メモリへのデータ通信（DMA 転送）

“B”:ノード間データ通信機能：FPGA メモリから別ノードのホスト CPU メモリへのデータ転送（提案手法）

“A” の様な、ノード内データ通信を実現する DMA 転送機能や複数 FPGA 間の直接通信機能については多くの研究がなされている。DMA 転送機能の先行研究として [11][12][5] などが、複数 FPGA 間の直接通信機能の先行研究として [10][9][8] などが挙げられる。FPGA と GPU、CPU を搭載したヘテロジニアスシステムでの FPGA から GPU へのデータ転送について、小林 等 [6] が研究を行っている。“B” の様な、FPGA メモリと別ノードのホスト CPU メモリとのデータ転送については先行研究が少ない。Markettos 等は、BlueLink と呼ばれるカスタマイズ可能な相互接続網のためのツールキットを提案している [11]。BlueLink は、FPGA と別のシステムを接続するための多種のトランシーバを抽象化するレイヤーで構成されており、ユーザはトランシーバや転送プロトコルを単一のユーザ API から利用可能となっている。評価項目は、10Gbps イーサーネット MAC のレイテンシと各トランシーバの回路面積であり、HPC 分野で利用されるような広帯域のトランシーバは議論されていない。

本研究報告では、FPGA メモリから別ノードのホスト CPU メモリへのデータ転送を実現する「ソフトウェアブリッジドデータ転送」機能を提案し、実効データ転送帯域の評価を行う。実装には、InfiniBand verbs と DMA 転送機能を利用し、それらをパイプライン的に動作させることで、経路全体のボトルネックである PCIe Gen3 x8 の理論ピーク値に対 81.0%の実効性能を達成した。

本研究報告は次の様な構成を取る。まず、第 2 節で評価に用いた、FPGA を搭載した予備評価システムとその DMA 転送機能について述べる。次節で、提案するソフトウェアパイプラインドデータ転送機能の詳細を述べる。第 3 節では評価結果を示し、議論を行う。最後に、第 4 節で報告をまとめを述べる。

2. FPGA 搭載予備評価システム

本研究報告の評価に用いた環境は、本番環境の FPGA 搭載クラスターの予備評価のために構築したもので、Preceding Evaluation Machine (PEM) と呼ばれる。図 3 に PEM とと転送部分のブロック図を示す。図中左の rNode は、ARM アーキテクチャの CPU を搭載し、FPGA は搭載していない。図中右の FPGA node は、x86 アーキテクチャの CPU を搭載し、PCIe 接続の FPGA ボード Intel Programmable Accelerator Card (PAC) with Arria10 を搭載している。各ノードの詳細を表 1,2,3 にまとめる。

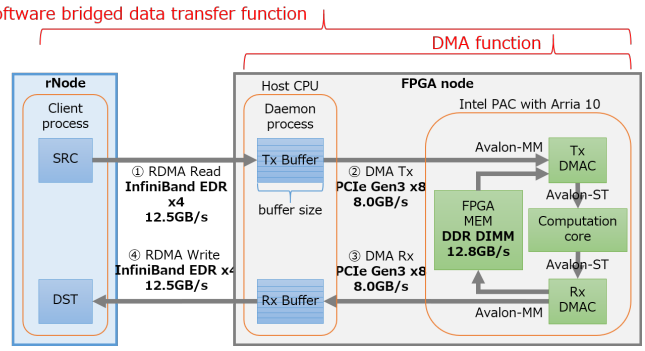

図 3: FPGA 搭載予備評価システム PEM と転送部分のブロック図：rNode と FPGA node は 100Gbps InfiniBand ケーブルで接続されている

OS	CentOS 7.6.1810
CPU	Cavium ThunderX2 CN9975 (28core @2.4GHz) x2
Memory	64GB x 2 = 128GB
InfiniBand NIC	Mellanox ConnectX-5 (MCX555A-ECAT[1])
libibverbs version	41mlnx1
InfiniBand driver	OFED.4.2.1.2.0.1.gf8de107.rhel7u4

表 1: rNode の構成

OS	CentOS 7.6.1810
CPU	Intel Xeon Gold 5122 (8core @3.60GHz) x2
Memory	48GB x 2 = 96GB
InfiniBand NIC	Mellanox ConnectX-5 (MCX555A-ECAT[1])
libibverbs version	41mlnx1
InfiniBand driver	OFED.4.2.1.2.0.1.gf8de107.rhel7u4
FPGA	Intel PAC with Arria10[3]

表 2: FPGA node の構成

FPGA	Intel Arria 10 GX (11AX115N2F40E2LG)
Logic elements	1150k
ALM	427200
Register	1708800
Emb. Mem. (M20K)	53Mb
DSPs	1518
Memory	Two channel of 4GB DDR4-2133
Interface	PCIe Gen3 x8, QSFP28

表 3: Intel PAC with Arria10 の詳細

Intel PAC では、FPGA の管理や外部 I/O のために FPGA のリソースが一部利用されている。Arria10 搭載 PAC の場合、ユーザが利用可能なリソースは次の通りであ

る。ALMs (Logic): 381213 (92%)、M20K (RAM Blocks): 51004 (94%)、Block memory bits: 55562240 (100%)、DSP Blocks: 1518 (100%)。

本研究報告では、FPGA node の x86 CPU は InfiniBand verbs を用いたノード間通信と DMA 転送を実行するために利用される。また、FPGA ボードへの回路データの書き込みや管理を行うが、計算リソースとしては利用しない。

PEM では図 3 の①〜④に示す 4 種類のデータ転送経路が存在する。本節では、予備評価として各経路の実行バンド幅の計測を行う。時間計測は *clock_gettime* 関数を利用して CPU 側で行い、10 回の試行の平均を取ったものを結果としている。*clock_gettime* 関数の第一引数を CLOCK_MONOTONIC とした。この設定では、レイテンシと解像度は 100 ns で、Linux 環境で最も高精度なタイマーとして知られている。

2.1 Direct Memory Access (DMA) 転送機能

ここでは、まず我々が実装した DMA 転送を行う API とハードウェアモジュールについて概説する。次に、ブロッキングデータ転送がどの様に行われるかを CPU メモリから FPGA メモリへのデータ転送を例に説明する。最後に、DMA 転送機能の評価について述べる。

2.1.1 DMA 転送機能の実装

DMA 転送機能は、C 言語で実装された API（含 制御ソフトウェア）と、FPGA に実装された Tx/Rx 用の 2 つの DMA コントローラ (DMAC) モジュールで実現される。DMA のユーザレベル API の名称は *afuShellDMA-Transfer()* であり、DMAC は Intel PAC に付属のサンプル *streaming_dma_afu* に機能を追加したものである。DMAC には Intel が提供している IP コア *modular scatter-gather DMA (mSGDMA)* を利用している。表 4 に Tx/Rx 用の 2 つの DMA コントローラ (DMAC) を合計したリソース使用量を示す。Intel PAC with Arria10 の全リソースの 1% 以下と、非常に小さいものとなっている。DMAC の論理合成と配置配線には、Intel Acceleration Stack 1.2 に付属の Quartus Prime Pro 17.1.1 for Intel PAC with Arria10 を利用している。

ALMs	Registers	BRAM Kbits	DSPs
2905	3055	135040	27

表 4: 2 つの DMAC の合計のリソース使用量

DMA API の内部実装について概説する。Tx と Rx の DMAC の性能を最大限に発揮するために、POSIX スレッドが各 DMAC の制御のために割り当てられている。また、図 3 の Tx Buffer, Rx Buffer に示す様な 8 エントリのリングバッファを用意している。2 つのリングバッファは、hugepage 機能と *fpgaPrepareBuffer()* を用いて、Kernel 空間上で連続したメモリアドレスとして確保し、CPU メモリから FPGA メモリへのデータ転送の高速化を実現している。また、リングバッファは CPU と FPGA 上の DMAC からアクセス可能なためゼロコピーが可能である。ゼロコピーは可能であるが、使いやすさの向上を目的として、ユーザデータを一旦リングバッファにコピーする様な形式を取っている。各エントリのバッファサイズは可変であり、サイズを変更した際の性能への影響を本節で述べる。表 1 は DMA 転送を行うユーザ API *afuShellDMATransfer* である。FPGA メモリを読み書きする場合は、読み出し・書き込み先アドレスに物理アドレスを指定する。

Listing 1: DMA 転送を行うユーザ API

```
1 fpga_result afuShellDMATransfer(
2   void* dst, // 書き込み先アドレス
3   const void* src, // 読み出し先アドレス
4   size_t count, // データ転送サイズ [byte]
5   dma_transfer_type_t type // データ転送方向
6   dma_handle_t dma_tx_h, // Tx ハンドラ DMAC
7   dma_handle_t dma_rx_h // Rx ハンドラ DMAC
8 );
```

CPU メモリから FPGA メモリへのデータ転送は次の様にして実現される。まず、ユーザ空間に存在する読み出し先アドレスのデータは、リングバッファのサイズに分割され、順にリングバッファの各エントリにコピーされる。次に、Tx DMAC が、各エントリのデータを PCIe バスと 512bit の Avalon-MM ポートを介して読み出す。読み出されたデータは Tx DMAC の書き出し側にある 512bit の Avalon-ST ポートに送られる。Tx DMAC の Avalon-ST ポートと Rx DMAC の Avalon-ST ポートは計算コアを介して接続されており、Tx DMAC から送られたデータは Rx DMAC へと流れる。Rx DMAC は流れて来たデータを Avalon-MM ポートを介して指定された FPGA メモリ上のアドレスへと書き出す。全てのデータが Rx DMAC の Avalon-MM ポートを通過した時に、Rx DMAC は完了信号として割り込みを発生させる。バッファサイズは可変であるが、サイズが小さい場合は、ユーザデータをコピーしたり DMAC へディスクリプタを書き込むなどの制御処理の回数が増えるため CPU 側の負荷が増加する。

FPGA メモリの理論転送帯域は、FPGA に実装された計算コアの動作周波数に 512bit を掛けたものとなる。本実装では、計算コアは 200 [MHz] で動作しているため、FPGA メモリの論理転送帯域は 200 [MHz] * (512 [bit] / 8 [bit/byte]) = 12800 [MB/s] となる。なお、本実装のホストメモリと FPGA メモリとのデータ転送の潜在的なボトルネックは、PCIe Gen3 x8 (7877 MB/s) である。

2.1.2 バッファサイズと実効データ転送帯域の評価

図 4 に、Tx/Rx バッファのサイズを 16KB から 8MB まで変更した際の CPU メモリから FPGA メモリへの実効

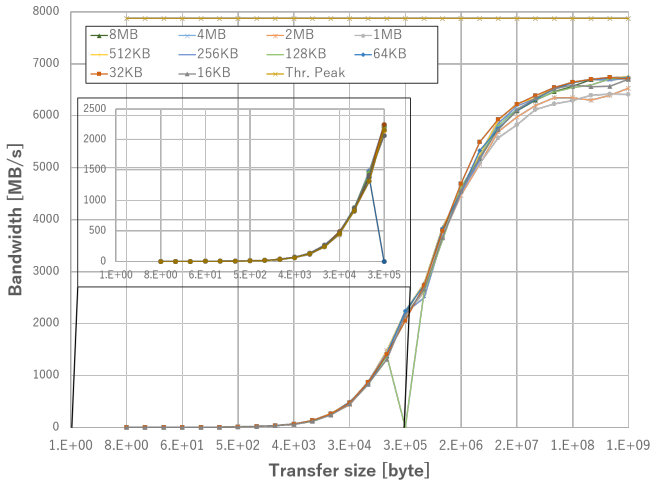

図 4: Tx/Rx バッファのサイズを 16KB から 8MB まで変更した際の、CPU メモリから FPGA メモリへの実効データ転送帯域

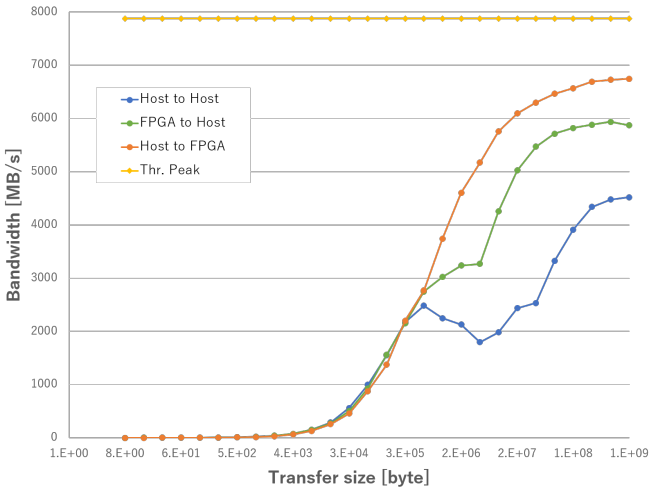

図 5: 3 種類の経路の DMA 転送機能の実効データ転送帯域：CPU メモリ →CPU メモリ、FPGA メモリ →CPU メモリ、CPU メモリ →FPGA メモリ

データ転送帯域の値を示す。また、図中の左上に転送サイズが 2 bytes から 256 Kbyte の時の値を拡大して示す。実装した DMA 転送機能はバッファサイズが 8MB で転送データのサイズが 1GB の時に、最大値である 6745.2 MB/s を達成した。これは PCIe Gen3 x8 (7877 MB/s) の理論ピーク性能の 85.6%である。また、バッファサイズが 256KB 以下の場合は性能が低下することもわかった。転送サイズが 1KB の際に、バッファサイズが 64KB の方が 8MB よりも 13%ほど高い帯域を記録している。全体としては、バッファサイズは性能に大きな影響を与えなかったが、多くの転送データのサイズで 8MB の時に最大値となることがわかった。また、FPGA node に搭載された CPU は、DMA 転送を行うのに充分な性能を持つものと考えられる。

上記の経路に加え、FPGA メモリから CPU メモリ、CPU メモリから FPGA を介した CPU メモリへの転送経路の実効データ転送帯域の測定を行った。前述の CPU メモリから FPGA メモリの評価と同様に、Tx/Rx バッファサイズを 16KB から 8MB まで変更してバッファサイズの性能への影響を評価した。どちらの転送経路も同様の傾向を示すことがわかった。つまり、バッファサイズの性能への影響は小さく、8MB の場合多くの転送サイズで最大値を達成するが、場合によってはバッファサイズが小さくとも最大値を達成する。図 5 は、バッファサイズが 8MB の場合の 3 つの転送経路の実効データ転送帯域を比較したものである。実装した DMA 転送機能は、転送サイズが 1GB の時に最大実効データ転送帯域を達成している。具体的には、FPGA メモリから CPU メモリへのデータ転送で 5872.6 MB/s (74.5 %)、CPU メモリから FPGA を介した CPU メモリへのデータ転送で 4521.3 MB/s (57.3 %) を達成した。なお、バッファサイズを 8MB 以上にした場合、実効データ転送帯域は変化しないか、場合によっては性能が低下していた。

FPGA メモリから FPGA メモリへの実効データ転送帯域は、転送サイズが 1GB の時に 12172.8MB/s (95.1 %) を達成した。詳細については、本論と直接は関係がないため割愛する。

ここからは、実装した DMA 転送機能と最新のオープンソース PCIe ライブラリである JetStream[12] との比較を行う。JetStream API はコピー DMA とゼロコピー DMA の両方に対応しているが、我々の実装と同じカーネル空間へのデータコピーを必要とするコピー DMA と比較を行う。JetStream のコピー DMA では、専用の 2 エントリのバッファを用いてダブルバッファリングが行われ、コピーと転送が同時に実行される。実行データ転送帯域は、CPU メモリから FPGA メモリへのデータ転送で最大 6.4 GB/s、FPGA メモリから CPU メモリへのデータ転送で最大 6.85 GB/s を達成している。なお、転送サイズの言及はない。CPU メモリから FPGA メモリへのデータ転送は我々の実装と同様の値だが、FPGA メモリから CPU メモリへのデータ転送は我々の実装より 15.9 %ほど高い値を達成している。我々の実装の長所は、JetStream よりも実効データ転送帯域の立ち上がりがかなり早い点が挙げられる。

2.2 InfiniBand RDMA Read/Write を用いたノード間データ転送機能

InfiniBand は HPC 分野における低レイテンシかつ高速なノード間データ転送を実現する相互接続網であり、InfiniBand verbs(IBv) は InfiniBand を利用するためのネイティブ API である。rNode と FPGA node のノード間データ転送には IBv を利用した。実際のデータ転送には IBv の RDMA Read と RDMA Write オペレーションを用いている。IBv でデータ転送を行うには、*ibv_reg_mr* 関数を

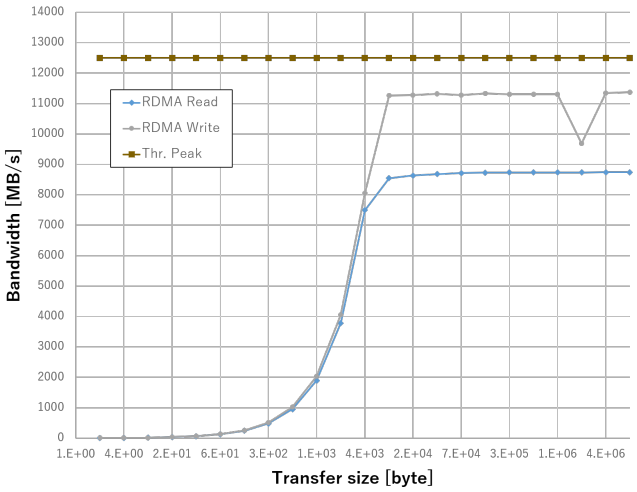

図 6: IBv の RDMA read と write を用いたノード間の実効データ転送帯域

用いて転送するデータを *protection domain* 内の *memory region* に登録する手続きが必要となる。また、データ転送以外の手続きとして、転送経路確立のために *queue pair* のやり取りが、転送の終了確認のために *completion_queue* の監視がそれぞれ必要となる。*memory region* への登録にかかる時間は、データサイズに比例して大きくなることが知られている。

予備評価として、IBv ライブラリに付属の *ib_read_bw* と *ib_write_bw* コマンドを利用して、RDMA Read/Write の実効データ転送帯域の評価を行った。IBv ライブラリ (libibverbs) のバージョンは 41mlnx1、HCA ドライバのバージョンは OFED.4.2.1.2.0.1.gf8de107.rhel7u4 であり、Maximum Transmission Unit (MTU) は 4096byte としている。図 6 に RDMA Read と Write の実効データ転送帯域と転送サイズの関係を示す。どちらも転送サイズが 8KB 以上で最大値を達成している。最大データ転送帯域は RDMA Read が 9162.1 MB/s (73.3 %)、RDMA Write が 11927.3 MB/s (95.2 %) であった。

3. ソフトウェア ブリッジド データ転送

本節では、我々が提案するノードを跨いで FPGA メモリへデータ転送を行う機能について述べる。以下、本機能をソフトウェア ブリッジド データ転送 (SBDT: software bridged data transfer) と呼称する。FPGA node のホスト CPU が CPU node ノードとのデータ転送と、FPGA メモリとのデータ転送をブリッジの様に繋ぐために、この様な名称となっている。ソフトウェアスイッチの様な役割とも言える。SBDT は IBv を用いたサーバ・クライアント型の通信モデルを採用している。第 2 節での評価から、PEM での SBDT のボトルネックは、ホスト CPU と FPGA ボードを繋ぐ PCIe Gen3 x8 であると予想される。

SBDT のために、3 に示すサーバ側の API が FPGA node のホスト CPU 側で要求の待受を行い、リスト 2 に示すクライアント側の API が rNode から所定のデータ転送を要求する。DMA API である *afuShellDMATransfer()* と同様に、FPGA メモリを読み書きする場合は、読み出し・書き込み先アドレスに物理アドレスを指定する。2 つの DMAC のハンドラは *dev_context_t* 構造体に内包されている。

Listing 2: クライアント側の SBDT 用 API

```
1 remote_fpga_result remoteAfuShellDMATransfer(
2 uint64_t src, // Destination address of data
3 uint64_t dst, // Source address of data
4 uint64_t tf_len, // Data transfer size [byte]
5 afu_shell_dma_transfer_type_t tf_type, // transfer type
6 remote_fpga_dma_handle_t dma_tx_h, // DMA handler
7 remote_fpga_dma_handle_t dma_rx_h // DMA handler
8 );
```

Listing 3: サーバ側の SBDT 用 API

```
1 void srv_afuShellDMATransfer(
2 // info. of FPGA and data transfer
3 dev_context_t* dev_context,
4 uint16_t req_size // Data transfer size [byte]
5 );
```

3.1 ナイーブ実装

SBDT のナイーブ実装を rNode の CPU メモリ (=*src*) を FPGA node の FPGA の計算コアを経て、rNode の CPU メモリ (=*dst*) へ戻す場合のデータ転送を例に説明する。以下、この転送経路を round trip と呼称し、図 7 にデータの流れを示す。データは、rNode から FPGA node の CPU メモリへ IBv RDMA Read を用いてノード間転送され、CPU メモリ上の受信用中間バッファに書き込まれる。次に、第節で述べた DMA API *afuShellDMATransfer* を用いて FPGA 上の計算コアを介して送信用中間バッファへ送られる。最後に、送信用中間バッファの内容が IBv RDMA Write を用いて rNode へ書き出され、転送が完了する。 図 8 にナイーブ実装の処理フローを示す。RDMA Read/Write を用いたノード間データ転送とそれに伴う制御処理、ノード内の DMA 転送が全て逐次的に行われる。また、サーバ・クライントの両プログラムの処理の詳細を以下に述べる。

(1) クライアントは、rNode 上の送信と受信のメモリ領域を IBv の *protection domain* 内の *memory region* として登録
(2) クライアントは、全データのサイズと *queue number* をサーバ側へ通知する
(3) サーバは、通知に従い所定のサイズの受信用バッファと送信用バッファをユーザメモリ空間に確保し、それを *protection domain* 内の *memory region* に登録する

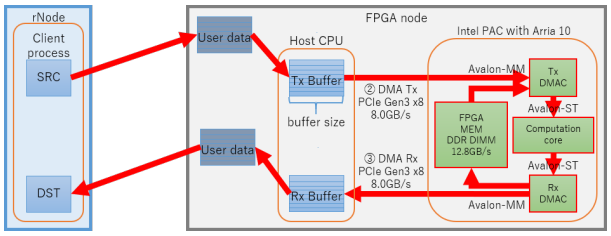

図 7: ナイーブ実装：全データが一度に転送され、各経路でのデータ転送も逐次的に行われる

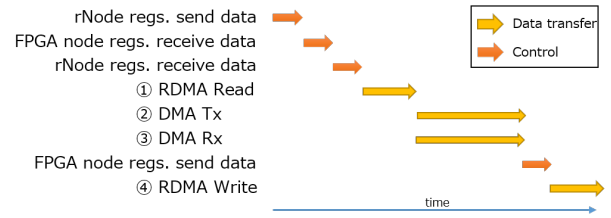

図 8: ナイーブ実装の処理フロー：転送とそれに伴う制御系の処理が逐次的に行われる

(4) サーバは、所定のサイズの RMDA Read リクエストを送り、クライアントが全データの送信を完了するまで *complete queue* をポールする
(5) データ送信が完了後、サーバは *afuShellDMATransfer* を用いて、計算コアを介した受信用バッファから送信用バッファへのデータ転送を行う
(6) サーバは、送信用バッファを *protection domain* に登録し、クライアントへデータサイズと *queue number* を通知する
(7) サーバは所定のサイズの RDMA Write リクエストを送り、クライアントへデータを送信する

round trip において 320MB のデータ転送を行った際の所要時間は次の様なものであった。なお、時間計測はサーバ側で *clock_gettime* 関数を利用した。データ転送以外の制御処理の所要時間は、(1) のクライアントでの送信と受信のバッファの確保と登録、通知に 0.121 秒、(3) のサーバでのバッファの確保と登録に 0.023 秒、(6) のクライアントでの登録と通知に 0.014 秒であった。また、実際のデータ転送の実効データ転送帯域と所要時間は、(4) の RDMA Read が 11187.5 MB/s で 0.027 秒、(5) の DMA 転送が 2103.9 MB/s で 0.144 秒、(7) の RDMA Write が 8979.1 MB/s で 0.034 秒となっており、充分な値が得られている。しかし、ナイーブ実装は上記の処理が全て逐次的に行われるため、全体の所要時間は 0.381 秒となり、実効データ転送帯域は 801.1 MB/s にとどまった。

3.2 全体のパイプライン化

より高い実効データ転送帯域を得るために、データ転送時の制御処理を最小限に抑え、各データ転送をパイプライン的に実行して遅延を隠蔽する。基本的なアイディアは、図 9 に示すような、ノード間データ転送に *afuShellDMA-*

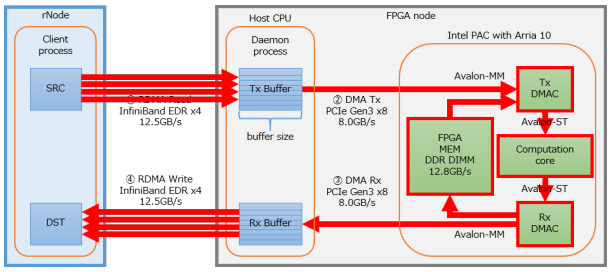

図 9: パイプライン実装：データが Tx/Rx バッファのサイズに分割されて転送され、各経路はパイプライン的に動作を行う

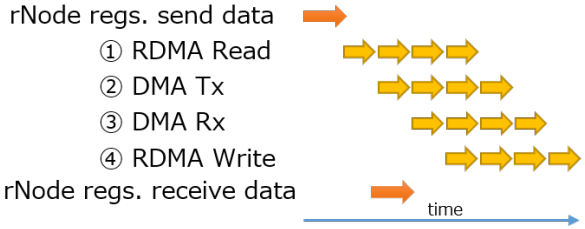

図 10: ナイーブ実装の処理フロー：転送に伴う制御系の処理は全て初期化時に行われ、データ転送のみがパイプライン的に行われる

Transfer の内部実装で用意された Tx/Rx バッファを直接利用し、バッファサイズ毎に各データ転送をパイプライン的に行うものである。具体的な処理を以下に述べ、図 10 にフローを示す。

(0) サーバは、起動時に Tx/Rx バッファの全エントリを *protection domain* 内の *memory region* として登録。バッファはサイズとアドレスがプログラム中で変更されないため事前に登録が可能である
(1) クライアントは、rNode 上の送信 (*src*) と受信 (*dst*) の領域全体を IBv の *protection domain* 内の *memory region* として登録。領域全体を登録すれば、その内部にある領域は *memory region* へ都度登録が必要ない
(2) クライアントは、全データを Tx/Rx バッファサイズに分割し、*queue number* と共に転送サイズをサーバに通知する
(3) サーバは、所定のサイズ（Tx/Rx バッファサイズ以下）の RMDA Read リクエストを送り、クライアントがデータの送信を完了するまで *complete queue* をポールする
(4) 各データの送信が Tx バッファのエントリへ受信され次第、サーバは Tx/Rx DMA を直接稼働させ、FPGA 上の計算コアを介した Tx バッファから Rx バッファへの DMA 転送を開始する。DMA の稼働には、ナイーブ実装で利用した *afuShellDMATransfer* ではなく、その内部実装を利用する
(5) 各データが Rx バッファへ格納され次第、サーバは RDMA Write リクエストを送り、クライアントがデータの受信を完了するまで *complete queue* をポールする

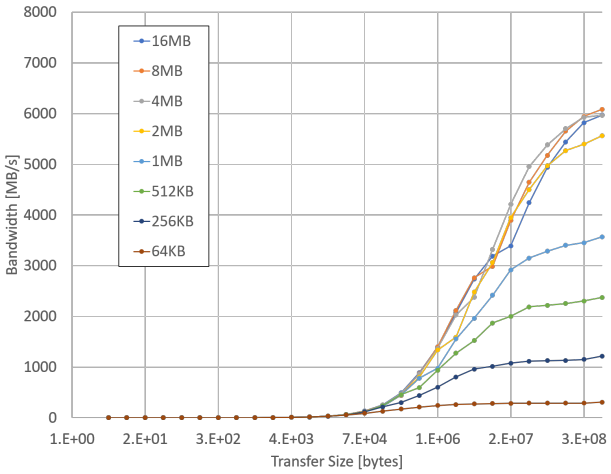

図 11: バッファサイズを 64KB から 16MB まで変化させた際の、rNode の CPU メモリから FPGA メモリへの実効データ転送帯域

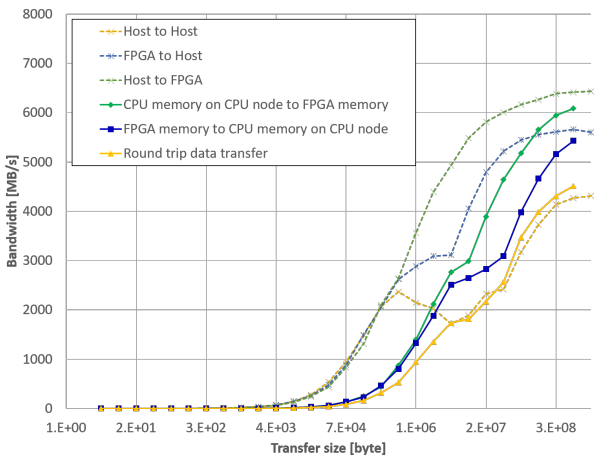

図 12: バッファサイズを 8MB とした時の 3 つの経路での実効データ転送帯域。比較のために、破線で DMA 転送の値を示す

(6) ステップ (3),(4),(5),(6) が全データの転送が完了するまで繰り返えす

3.3 評価と議論

ここでは、パイプライン実装の転送サイズと実効データ転送帯域を関係を測定する。第 2.1.2 節と同様に、Tx/Rx バッファのサイズを 64 KB から 16 MB まで変更した際の性能への影響も評価した。バッファサイズが 32KB 以下の時は転送に失敗したため、測定項目から除外している。rNode の CPU メモリから FPGA メモリへの実効データ転送帯域の測定結果を図 11 に示す。本転送経路のボトルネックは、PCIe Gen x3 (理論ピーク帯域: 7877MB/s) である。バッファサイズが 1MB 以下の場合、転送帯域はピーク帯域に対し大幅に低い。これは、ノード間データ転送に対し、DMA 転送が遅いためにパイプラインがストールしてしまったためであると考えられる。図 4 と図 6 の測定結果から、転送サイズが 1MB の場合は、RDMA Read が DMA 転送よりも実効データ転送帯域が高いことが裏付けとなる。バッファサイズが 2MB 以上の場合は、実効データ転送帯域はピーク帯域の 70〜80 %を達成している。バッファサイズが 8MB の場合、最大値を達成している。これはパイプラインがストールせずに効果的にデータ転送が行われているためであると考えられる。図 4 と図 6 の測定結果から、転送サイズが 8MB の場合は、RDMA Read と DMA 転送の実効データ転送帯域が近い値であることが裏付けとなる。本データ転送経路以外の、FPGA メモリから rNode の CPU メモリ、round trip も同様の傾向を示した。

図 12 に 3 つの経路の実効データ転送帯域をまとめる。実線で表されたグラフが SBDT の値であり、破線で表されたグラフは比較のためにプロットした DMA 転送の値である。各経路は転送サイズが 512MB の時に転送帯域が最大値となった。転送帯域の最大値とピーク性能に対する比率は、rNode の CPU メモリから FPGA メモリでは 6379.1 MB/s (81.0 %)、FPGA メモリから rNode の CPU メモリでは 5696.1 MB/s (72.3 %)、round trip では 4732.0 MB/s (60.1 %) となった。CPU メモリから FPGA メモリへの DMA 転送と、rHost のメモリから FPGA メモリへの SBDT 転送を比較すると、後者の方がグラフの立ち上がりが遅い。これは、SBDT 転送がノード間転送の時間を完全には隠蔽できていないためである。さらなる性能向上の案として、ステップ (3) で複数の RDMA Read リクエストを一度に送ることが考えられる。

4. 結論

本研究報告では、PCIe 接続された FPGA ボード上のメモリから別ノードのホスト CPU メモリへデータ転送を行う、ソフトウェア ブリッジド データ (SBDT: software bridged data transfer) 転送の提案を行った。評価は我々が FPGA クラスターの予備評価目的に構築した Preceding Evaluation Machine (PEM) 上で行った。予備評価として、FPGA node 内の CPU メモリから FPGA メモリへの DMA 転送機能の実装と実効データ転送帯域の評価、InfiniBand を用いたノード間の実効データ転送帯域の評価を行った。評価に際し、SBDT 転送と DMA 転送に利用する Tx/Rx バッファのサイズを 16 KB から 8 MB に変化させて、転送サイズと実効データ転送帯域の関係を評価した。その結果、バッファサイズが 8MB で転送サイズが 1GB の時に、DMA 転送では、CPU メモリから FPGA メモリへのデータ転送で 6745.2 MB/s (85.6%)、FPGA メモリから CPU メモリへのデータ転送で 5872.6 MB/s (74.5 %)、CPU メモリから FPGA を介した CPU メモリへのデータ転送で 4521.3 MB/s (57.3 %) を達成した。また、バッファサイズが 8MB

で転送サイズが 512MB の時に、SBDT 転送では、rNode の CPU メモリから FPGA メモリで 6379.1MB/s (81.0 %)、FPGA メモリから rNode の CPU メモリで 5696.1 MB/s (72.3 %)、round trip で 4732.0 MB/s (60.1 %) となった。SBDT 転送は DMA 転送に比べ、実効データ転送帯域の立ち上がりが遅い。これは、SBDT 転送がノード間転送の時間を完全には隠蔽できていないためであると考えられる。

今後の展開として、SBDT 転送のさらなる実効データ転送帯域の向上と、レイテンシの詳細な測定、複数 FPGA node への拡張である。

5. 謝辞

本研究の一部は、JSPS 科研費 19K20282 の助成を受けたものです。

参考文献

[1] : ConnectX-5 VPI Adapter Cards User Manual - ConnectX-5 InfiniBand VPI - Mellanox Docs, https://docs.mellanox.com/display/ConnectX5IB.

[2] : Cygnus Supercomputers Center for Computational Sciences, Tsukuba University., https://www.ccs.tsukuba.ac.jp/eng/supercomputers/#Cygnus.

[3] : Intel Programmable Acceleration Card with Arria 10 GX FPGA, https://www.intel.com/content/www/us/en/programmable/products/boards_and_kits/dev-kits/altera/acceleration-card-arria-10-gx.html.

[4] Caulfield, A. M., Chung, E. S., Putnam, A., Angepat, H., Fowers, J., Haselman, M., Heil, S., Humphrey, M., Kaur, P., Kim, J., Lo, D., Massengill, T., Ovtcharov, K., Papamichael, M., Woods, L., Lanka, S., Chiou, D. and Burger, D.: A cloud-scale acceleration architecture, *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 1–13 (online), DOI: 10.1109/MICRO.2016.7783710 (2016).

[5] Jacobsen, M. and Kastner, R.: RIFFA 2.0: A reusable integration framework for FPGA accelerators, *2013 23rd International Conference on Field programmable Logic and Applications*, pp. 1–8 (online), DOI: 10.1109/FPL.2013.6645504 (2013).

[6] Kobayashi, R., Oobata, Y., Fujita, N., Yamaguchi, Y. and Boku, T.: OpenCL-ready High Speed FPGA Network for Reconfigurable High Performance Computing, *Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region*, HPC Asia 2018, New York, NY, USA, ACM, pp. 192–201 (online), DOI: 10.1145/3149457.3149479 (2018).

[7] Plessl, C.: Bringing FPGAs to HPC Production Systems and Codes, *H2RC'18 workshop at Supercomputing (SC'18)*, (online), DOI: 10.13140/RG.2.2.34327.42407 (2018).

[8] Sanaullah, A. and Herbordt, M. C.: FPGA HPC Using OpenCL: Case Study in 3D FFT, *Proceedings of the 9th International Symposium on Highly-Efficient Accelerators and Reconfigurable Technologies*, HEART 2018, New York, NY, USA, ACM, pp. 7:1–7:6 (online), DOI: 10.1145/3241793.3241800 (2018).

[9] Sanchez Correa, R. and Pierre David, J.: Ultra-low latency communication channels for FPGA-based HPC cluster, *Integration*, Vol. 63 (online), DOI: 10.1016/j.vlsi.2018.05.005 (2018).

[10] Tarafdar, N., Lin, T., Fukuda, E., Bannazadeh, H., Leon-Garcia, A. and Chow, P.: Enabling Flexible Network FPGA Clusters in a Heterogeneous Cloud Data Center, *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '17, New York, NY, USA, ACM, pp. 237–246 (online), DOI: 10.1145/3020078.3021742 (2017).

[11] Theodore Markettos, A., Fox, P. J., Moore, S. W. and Moore, A. W.: Interconnect for commodity FPGA clusters: Standardized or customized?, *2014 24th International Conference on Field Programmable Logic and Applications (FPL)*, pp. 1–8 (online), DOI: 10.1109/FPL.2014.6927472 (2014).

[12] Vesper, M., Koch, D., Vipin, K. and Fahmy, S. A.: JetStream: An open-source high-performance PCI Express 3 streaming library for FPGA-to-Host and FPGA-to-FPGA communication, *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, pp. 1–9 (online), DOI: 10.1109/FPL.2016.7577334 (2016).